# IMAGE PROCESSING FOR URBAN SCALE ENVIRONMENTAL MODELLING

Paul Richens
The Martin Centre for Architectural and Urban Studies
University of Cambridge
6 Chaucer Road, Cambridge, CB2 2EB, UK

## ABSTRACT

If a map of a city is encoded as a Digital Elevation Model, it becomes amenable to image-processing software, such as the public-domain NIH Image application. Standard techniques can be used to measure plan areas and volumes and simple macros can be devised to measure perimeter length and wall areas. A macro for calculating shadow volumes is elaborated for the simulation of solar gains and daylight, including indirect lighting, leading to the possibility of an image-based urban-scale environmental model.

## INTRODUCTION

Two years ago, in 1995, a group of colleagues at The Martin Centre for Architectural and Urban Studies at the University of Cambridge started an investigation of Zero Emission Urban Development, known as Project ZED. Fundamental to this project was the investigation of urban texture, and its relationship to micro-climate, environmental performance and energy use. The energy investigation was informed by earlier work by Baker and Steemers[1985] which had led to the "LT Method" for estimating the energy needs of commercial buildings. LT is a simple manual or spreadsheet method which requires the user to split a building into active (deep plan, artificially lit and ventilated) and passive (naturally lit and ventilated) zones, and further to categorise the passive zones by orientation. Once these zones are identified and measured, LT provides a series of graphs and procedures to arrive at the annual estimated primary energy consumption, taking into account solar gain and artificial lighting, as well as the usual fabric, ventilation and casual gains and losses. The procedure is simple to use, but requires manual measurement of the building, so is hardly applicable to the urban scale. One of the long-term aims of the present work is to find a corresponding methodology that can be applied to substantial areas of cities, without demanding detailed measurement of individual buildings. Ideally it will be more sensitive than the original LT Method to the surrounding context; the over-shadowing, shelter, noise and pollution that characterise a city micro climate.

The issue of texture, and how to characterise it in a way that would correlate with micro climate, is relatively unexplored. Webster[1995] derived some textural signatures from satellite images, which he found to correlate with residential density. Project ZED was to focus on three study areas, each of 400 metres square, in three different European cities; London, Toulouse and Berlin. Early on, the team prepared black-and white "figure-ground" maps of their study areas, and later, when height information became available, made foam block models, for use in the wind-tunnel and artificial sky. Could image processing provide a third line of attack?

The primary tool for texture analysis is the two-dimensional Fast Fourier Transform (FFT); it seemed interesting to scan the figure-ground maps, and perhaps to add the height information to them, forming what the geographers call a Digital Elevation Model (DEM). But first we needed some software. A search of the world-wide web using the keywords "Mac", "image" and "FFT" produced three pieces of download-able Macintosh software with the capability of performing a 2D FFT. One was a tutorial on the use of the FFT for filtering, and another was aimed at applications in astronomy. The most promising was a public-domain application from the US National Institute of Health, called NIH Image, designed for the analysis and measurement of medical images, chiefly microscope slides, but also used for X-rays, CT and MRI scans. In the end, the FFT was of little importance, but the other capabilities of NIH Image turned out to be extraordinarily adaptable to our purpose.

## NIH IMAGE

This section will introduce the basic concepts and terminology of image processing, by reference to NIH Image. A complete description can be found in the download-able manuals, (see bibliography). For a full treatment of image processing, see Jain[1989]. Many of the ideas in NIH Image will be familiar from popular programs such as Adobe Photoshop; the difference is one of emphasis. Photoshop is concerned mainly with visual appearance, whereas NIH Image concentrates on measurement and analysis.

The basic data structure is the image, a 2D array of pixels, each storing a single byte (8 bits), typically understood as an integer in the range 0..255. Closely associated with the image, is its LUT (Look up table), which translates each of these integers into a colour that is displayed on the screen. The LUT can be freely changed, changing the appearance of the image, without damaging the underlying data stored in each pixel. Image and LUT can be written to disk as a unit, typically as a TIFF file. Images usually originate with a scanner; video frames can be captured directly if the computer has an AV board. Images can be originated or edited on-screen using the usual paint-box tools for line-drawing, selection, fill, brushing and erasing.

An extra piece of information stored with each image is its scale and calibration. Scale defines the width of the pixel, while calibration allows the 0..255 value of each pixel to be mapped onto some physical variable of interest, such as optical density. This mapping need not be linear. With this information Image can display cross-sections along any line you draw on the screen, or produce an axonometric rendering of the whole image. It is also possible to display thick sections, by drawing a thick line. The value displayed is the average, taken through the thickness of the section.

A large part of the functionality is aimed at image enhancement, using two major techniques. The first is LUT modification, which can alter the brightness and contrast, or even apply false colour to the pixel values. Thresholding turns the grey-scale image into a black and white one, with interactive control of the threshold value. Density slicing turns a narrow range of pixel values bright red; again with interactive control. Dragging the slice through the range gives a precise way of exploring the values in the image. Both thresholding and density slicing are performed by LUT modification, but it is possible to force the changes into the image itself, which then becomes a binary image with values of 0 (false, white) and 255 (true, black) only.

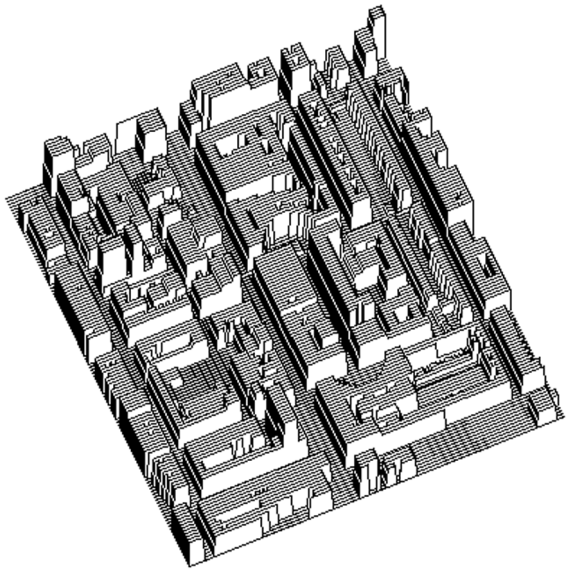
The second method of image enhancement is by applying filters. The fundamental idea behind image filters is the kernel, a small window or region of pixels (say 3 by 3 or 5 by 5). The filter operates by centring the kernel on each pixel in the image in turn, and performing some computation on the values seen under the kernel. The central pixel is replaced by the result of the computation. So for example the Max filter replaces each pixel by the maximum value in its 3 by 3 neighbourhood, and the Median filter replaces each pixel by the median (central) value of its neighbourhood (which is a popular way of removing noise from an image).

Convolution filters work by forming a weighted average of the pixels under the kernel; the kernel cells specify the weights. If the weights are all positive, the result is some sort of smoothing or blurring. More sophisticated kernels (with some negative weights) can sharpen an image, differentiate it, or detect edges.

NIH Image provides some all-important features for doing arithmetic on complete images, for example by adding or multiplying corresponding pixels, or taking the maximum or minimum values. The result is a complete new image. In general these operations are computed at higher precision, scaled and offset by a stated amount, and then clipped to the 0..255 range allowed in each pixel.

NIH Image will measure any selected region of an image, reporting how many pixels it contains, their mean value, and standard deviation. It will also draw a histogram.

In addition to these widely useful features, NIH Image provides a number of more specialised algorithms, such as background removal (to compensate for the uneven illumination of a microscope slide), erosion and dilation, edge detection, and the FFT itself. But the most important is the macro capability, which allows users to code their own commands, using a simplified form of Pascal. The interpreted macro language has full access to all the menu commands, and to a number of lower-level functions. It is possible, but slow, to operate at the level of individual pixels, but preferable to use commands, such as image arithmetic, which are hundreds of times faster.



*Digital elevation model*

## DIGITAL ELEVATION MODELS OF CITIES

The DEMs of the three cities studied in Project ZED started as hand-drawn figure-ground maps prepared from published maps. These were scanned into TIFF files and loaded into NIH Image. A suitable scale for height information was chosen (such as 10 units per metre, maximum height 25.5m). Height information (from a field survey) was added using painting tools, chiefly area-select and flood-fill. Each image was calibrated to fix the dimension of a pixel as measured on the ground, and the size in metres of one unit of elevation data. Typically a study area of 400m square would be prepared as a 512 by 512 pixel image (the FFT algorithm requires images to be square, and a power of two in side), but for many experimental purposes a downsized image 256 pixels square proved to be adequate in resolution, and around four times faster to process.

With this simply constructed DEM, NIH Image was able to measure most of the usual built form statistics. Histograms of building height, arbitrary cross-sections, and axonometric representations came directly. A very simple macro to measure the whole image, and multiply the number of pixels by the mean value (with suitable calibration corrections) gives the built volume. By thresholding the image at value 1, and re-measuring, we obtain the built area. This works because the ground profile in the study areas is essentially flat, and the ground is at level zero throughout. If this were not the case we could use the background removal feature to find ground level, which could be subtracted from the image before measuring areas and volumes. By using thresholding to extract an image at any particular level, we can repeat area measurements at any horizontal cross-section.

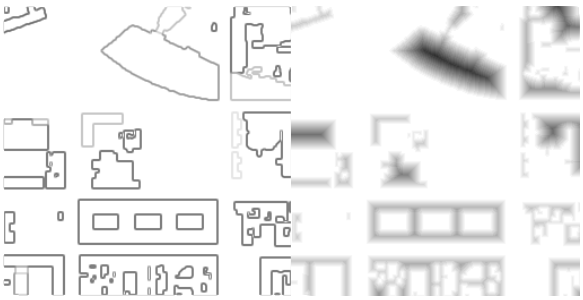
Measuring external wall area (or the thresholded equivalent, perimeter length), proved more challenging. There is a command to extract edge pixels from the image, which seems a good start. However, summing them and multiplying by the scale does not necessarily give the right answer. Walls aligned to the axes measure correctly, but walls at 45 degrees measure short, by a factor of $\sqrt{2}$.

A more accurate answer can be obtained by using a standard image-processing technique, the Sobel edge-detector. The image is convolved separately with two kernels:

```
 1  2  1      -1  0  1
 0  0  0      -2  0  2
-1 -2 -1      -1  0  1
```

The first constructs the derivative in the *y* direction $f_y$, the second $f_x$. These derivatives are equal to eight times the projected area of the wall in each direction. The total area is therefore obtained by forming the square-root of the sum of the squares of the two derivatives. NIH provides a command "Find Edges" which does the whole operation, though care is needed (in pre-scaling the images) to make sure that overflow in the limited precision arithmetic it uses does not cause clipping in the result. If the resulting edge image is examined carefully it will be seen that the edges are 2 pixels wide. Whereas the original edge between a high and a low elevation value apparently occurred in the crack between two pixels, it is now smeared out, occupying all the pixels adjacent to the boundary. This smearing of edges, so that they now occupy some actual pixels (rather than the cracks between them), turned out to be useful in other contexts, as it gives us some location at which to store other information relating to edges, such as solar irradiation.

If the convolutions are carried out separately, then it is possible to use the results to classify edges by their orientation computed as $atan(f_y/f_x)$.



*Wall area and plan depth displays*

Deep-plan areas (at a certain distance from the perimeter) can be displayed using density-slicing to extract the level at which to work, and then eroding the image to leave the deep-plan area. The most recent version of NIH Image has a more sophisticated command, which will code pixels by their depth from the edge; this allows us to build a histogram of areas sorted by depth from the facade.

## TEXTURE ANALYSIS

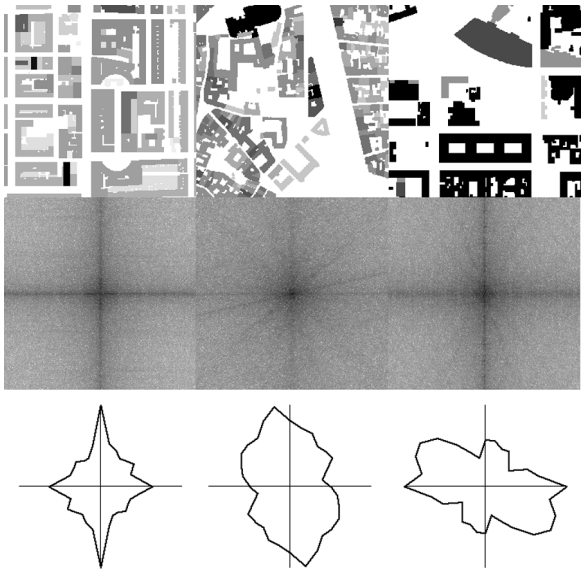
One aspect of texture that interested the Project ZED team was its directionality. The study areas are not isotropic; some wall alignments and street orientations are preferred over others, with likely consequences for noise, wind and pollution movement. We were interested in finding some measure of this directionality that could be correlated with the results of wind-tunnel experiments on the rate of clearance of smoke from models of the study

area, under different flow orientations (NIH Image, incidentally, proved very effective in analysing the video-tapes of these experiments).

The prime tool for studying periodicity and orientation in images is the two-dimensional FFT. The results of applying it to our three study areas are shown below. Although the primary orientations are distinguished in these plots, they are, in general, hard to interpret.

An alternative, more physical, measurement, was proposed by my colleague Dr Baker. Called Directional Porosity, he proposed drawing thick cross-sections of varying orientations, averaging the height across the section, and then measuring the variance along it. A section perpendicular to the street pattern would show a strong pattern of troughs where the streets occur and have a high variance, whereas a section oblique to the streets would have a much lower variance.

The basic operation required for this analysis, averaging values over a deep cross-section of arbitrary orientation, is a standard command in NIH Image. So it was quite simple to construct a macro which iterated over 16 or so orientations, and for each defined a thick cross-section, measured its profile, and computed its variance. Finally, the results were plotted as a polar diagram (Bakergram) showing how Porosity varies with orientation.



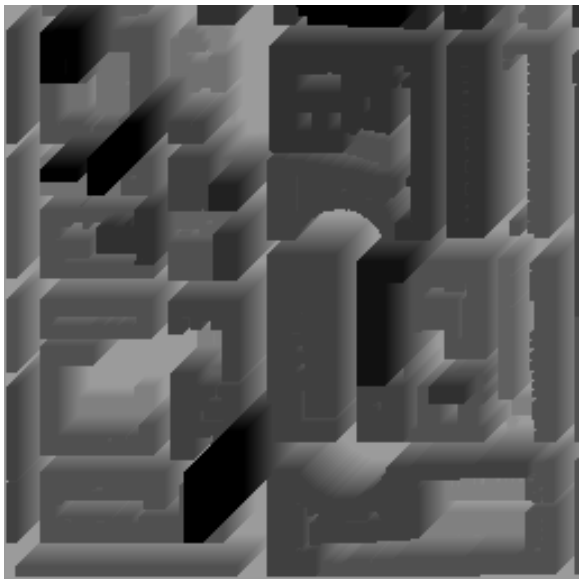
*Three cities, with Fourier transforms and Bakergrams*

## SIMULATIONS

The results obtained so far would provide some of the basic inputs to an urban version of the LT Method, such as volume, wall and floor areas; location of shallow and deep plan zones; and the orientation of walls. The next issues to tackle are daylight and sunlight; and as the urban geometry is available, it should be possible to deal with overshadowing and obstruction in a very precise way.

The central algorithm is one to compute shadows for an arbitrary angle of lighting. The approach taken is to compute shadow volumes as a DEM, that is the upper surface of the volume of air that is in shadow. We start by defining the three components of the vector pointing towards the light. Then we compute the components of an opposite vector, scaled so that the larger of the *x* and *y* components is just 1 pixel, and the *z* component is adjusted to the image calibration. If we translate the DEM by the *x* and *y* components, and simultaneously reduce its height by subtracting the *z* component, we get part of the shadow volume. If we continue translating and lowering by multiples of this vector, and take the union of this volume with that previously calculated, we build up the whole shadow volume. The process can be stopped when all levels are zero, or the translation has shifted the volume right off the image.

This is easily encoded as an NIH Image macro. The translation is done by cutting and pasting into a shifted selection area, and the union operation is done by computing the maximum of two images. The results are shown below, and the actual code in the Appendix.



*Shadow volumes*

To reduce the shadow volume to an actual map of shadows on the roofs and ground of the city, the original DEM is subtracted from the shadow volume. Pixels with zero value are in light (negative values are clipped to zero), positive values are in shade.

This calculation computes shadows for an arbitrary lighting angle; it is straightforward to add a procedure to calculate shadows from the sun for any given latitude, time of year, and time of day, using the usual astronomical formulae. The next stage is to calculate solar irradiation, the actual amount of energy received at each pixel, from the sun, taking into account the solar altitude, angles of incidence, and shadowing.

Computing the clear-beam normal surface irradiation can follow a standard method – we used that in Page[1986]. Shadowing uses the previous algorithm. It remains to find a way of computing the effects of the angle of incidence.

Horizontal ground and roof surfaces do not present much of a problem; each pixel has a known area, and the angle correction depends simply on the sine of the solar altitude. But wall surfaces receive a good deal of energy, and they are not even represented in the DEM. The answer is to use the same filters as in the Sobel edge-finding algorithm. The $f_x$ and $f_y$ derivatives can be interpreted as the projected area of the pixel in each direction. The projected area in the vertical direction is simply the area of the pixel, and so constant for the whole image. At each pixel we have a vector $(-f_x, -f_y, 1)$ which is parallel to the surface normal at that pixel, and whose magnitude is equal to the area of surface, which can be quite substantial at edges. The irradiation is found by taking the dot product with the vector to the sun, which amounts to scaling two images by two constants, adding them together, and adding a third constant. Negative values are removed, as they correspond to surfaces facing away from the sun, and all those in shadow. The remaining values, which give for each pixel its area multiplied by the cosine of the angle of incidence, are multiplied by the direct-beam normal irradiation, and that is the answer.

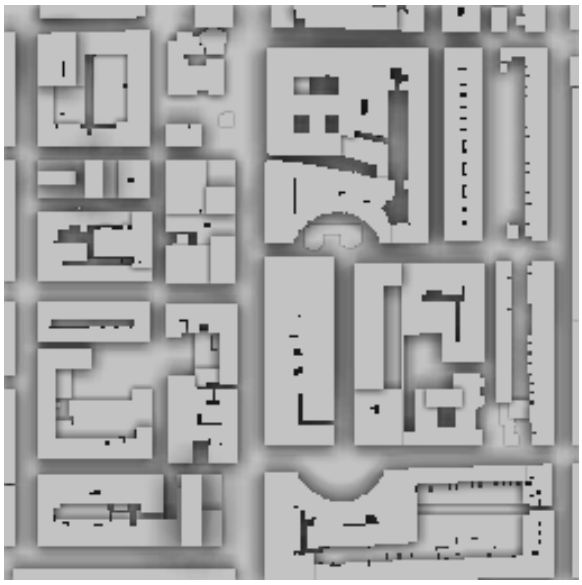
A defect of this calculation is that the edge pixels have levels associated with either the adjacent ground, or the adjacent roof. If a wall is partly in shadow, exactly half of the pixels will receive energy, regardless of whether the shadow covers 10% or 90% of the height. A better result can be obtained by using a pair of levels for comparison with the shadow volume. One level is derived from a 3 by 3 max filtering of the DEM, the other from a min filter. These two levels will differ from each other in all the edge pixels. By determining the height of the shadow as a proportion of the interval between these two levels, a correct allocation of irradiation can be made.

Having computed the irradiation for a given moment, it is simple to integrate over a day or longer, to compute the solar energy available. This could be used for a thermal model, to assess sunshine availability for planting, or for planning solar cell installations.

*Solar irradiation over one day at equinox*

The basic shadow algorithm can also be used as the basis for an evaluation of direct daylight. A particularly simple case is the calculation of the sky-component of daylight factor on all the exposed horizontal ground and roof surfaces of the city. We simply compute the shadows for a large number of light sources, distributed over the sky, and for each pixel, count the number of times they are in light. So if we use 255 samples, any pixel whose count is 255 can see all the sky and has 100% sky component, while a count of 0 means that it cannot see the sky at all.

To get meaningful results, it is necessary to distribute the sample points over the sky in the correct manner. If a uniform distribution is used, then what is measured is the solid-angle of sky visible from each point. While this may be useful, it is not the usual measurement used in daylight analysis. However, if we distribute our points evenly over a unit circle in the horizontal plane, and then project up to a unit hemisphere, we obtain a cosine-weighted distribution which is correct for computing sky-component for a horizontal plane, assuming a uniform sky luminance.



*Sky component on horizontal plane, exterior.*

The prediction of daylight in the open air is nothing like so important as the same calculation carried out for interior space. Again, the computation of direct sky component for interior space at a given working plane level is surprisingly easy. We assume a 100% transparent facade, and need to know the level of the ceiling, or whatever defines the upper cut-off angle, in addition to that of the working plane.

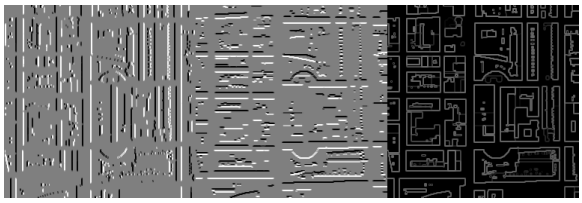
The algorithm proceeds as before, by casting shadows from random points appropriately distributed across the sky, but with a slight modification. Each shadow volume is sliced at the level of the ceiling, and the resulting shadows translated backwards along the light vector until they reach the working plane level. Then the pixels in light are counted as before.

It is possible to elaborate this to compute daylight at many levels simultaneously, with little increase in time, by slicing each shadow volume at a number of different levels. In this way we can build up a picture of daylight availability at many different levels, rather like a medical tomograph, for the cost of 255 applications of the shadow-casting algorithm.

These daylight and sunlight algorithms have simulated only the direct light from the sun or sky, and have done nothing about reflected light. This is certainly inadequate for architectural interiors, and arguably so for exteriors as well. In southern European cities, streets are often deep, and see little sky. Indirect light can be very important, but it is hard to calculate[Sillion 1994]. Existing radiosity or stochastic ray-tracing software struggles to deal with a single room, let alone several hectares of city.

However, it does seem possible to modify some of the ideas developed earlier to calculate indirect light, using image processing effectively to cast rays from thousands of pixels simultaneously. If we repeat this a few hundred times, using random angles for the rays, this time distributed over the whole sphere, we begin to get a useful answer.

As a preliminary, we need to know the unit surface normal at each pixel. In principle this is easy; we use the Sobel filters to calculate the $f_X$ and $f_Y$ derivatives, which can be used to construct two tangent vectors $(1, 0, f_X)$ and $(0, 1, f_Y)$. The normal is their cross-product $(-f_X, -f_Y, 1)$, which has then to be normalised by dividing by $\sqrt{(f_X^2 + f_Y^2 + 1)}$. This last is rather beyond the capability of image arithmetic commands, so a macro was prepared which computes the derivative images, assembles them side by side into a single image, which is then read row by row, normalised, and replaced. Finally the result is broken up into three separate images.



*Three components of the normal vector*

The three images which provide the components of the normal are used, once a ray orientation has been chosen, to compute the cosine of the angle between the ray and the normal, by means of a dot product. The cosines form yet another image, with positive values looking along the ray, and negative ones looking back.

The Sobel filters broaden edges into a width of two pixels. To reflect this in the DEM we apply a 3 x 3 box filter (one with all the weights equal to 1/9). This has no effect in areas of constant level, but at edges performs a linear interpolation. Walls become two pixels wide, with values of 1/3 and 2/3 of the height.

To start the process, we create another image to hold initial radiance values. This can be initialised by selecting a sun position, casting shadows, and for every lit pixel setting a radiance value proportional to its cosine, provided it is positive (actually the check is redundant, as negative cosines belong to back-facing pixels, which must be in shadow).

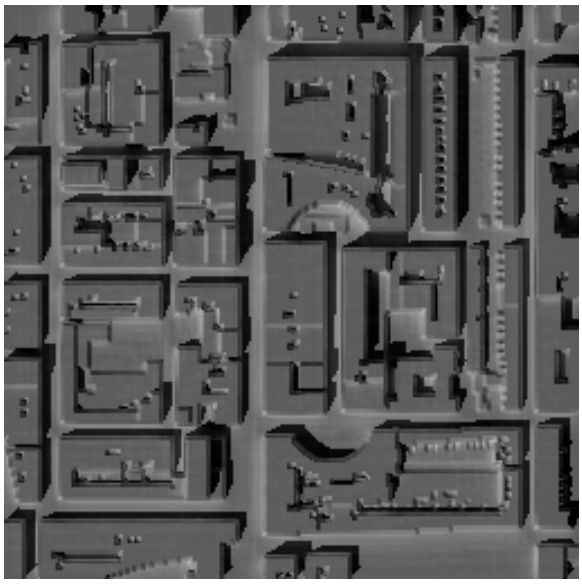
The process of distributing light from these illuminated pixels will be repeated many times, at random orientations. For each orientation, we compute the cosine array. Negative values are

potential transmitters of light. Positive cosines correspond to pixels that are facing the right way to receive light.

The process of tracing the rays from receiver to transmitter is similar to, but more elaborate than, the algorithm for shadow volumes. Imagine the transmitting DEM being shifted back along the ray, changing in level as it goes. At each step, we check to see which pixels are now below the level of the unshifted receiver DEM. A pixel that goes below represents a ray that has been intersected, so we find the corresponding shifted radiance value, multiply by the receiver's cosine and accumulate as part of the incoming irradiance.

On each pass, a pixel may receive irradiance from only one other, which must be the first one encountered in traversing the ray. This is handled by setting up a binary mask image, initially true for every potential receiver. The mask is turned to false as soon as any light has been received, and is used to inhibit light being delivered from any further intersections that may occur. The final state of the mask shows which pixels see light from the sky.

After many rays have been cast, the accumulated irradiance is multiplied by $r/\pi n$, where r is the reflectance, and n the number of rays, to yield the extra radiance at that pixel due to the first-order reflected light. The whole process should then be repeated (using the extra radiance at the transmitters) until the amount of light shifted becomes insignificant.



*Direct sunlight, and first order reflected light*

## DISCUSSION

To one accustomed to traditional computer models of buildings, based on points, lines and surfaces, these algorithms are strikingly simple and efficient. The central shadow volume algorithm (see Appendix) is less than a page of code, and took only a few hours to develop. To do the same sort of thing in a traditional modelling environment is likely to take weeks. Yet the image processing algorithm can cast shadows for a 512x512 pixel image of 16 hectares of London in under 10 seconds (or around 2 at 256 pixel resolution), running on a laptop Macintosh. Unlike traditional methods, the speed does not depend on the complexity of the scene, only on the size of the image. This performance is simply astonishing to anyone familiar with ordinary rendering software.

The image-processing commands effectively give the macro programmer a parallel computer for doing simple operations to thousands of pixels at the same time. So the algorithms presented here are essentially parallel, and quite different in structure to conventional ray-casting algorithms, where most of the effort is detecting cases where processing can be avoided. In image-processing, it is simpler and faster just to do the operation.

Of course, the speed is obtained at the cost of precision. Our images are at a resolution of around 1 metre to the pixel. But it can be argued that this is a highly appropriate resolution for urban-scale investigations, and it is a strength of this approach that it can capitalise on this simplification of the problem.

There are several areas where accuracy might be problematical. For example the edge-finding technique uses 3x3 kernels, so it cannot respond to very fine distinctions in orientation. A larger kernel would be more discriminating, but would smear edges into broader bands. This would have some advantages, but only if the images were at finer resolutions. The performance of the algorithms is likely to be $O(n^3)$ with the linear size of the image in pixels. The basic operations are $O(n^2)$ ie proportional to the number of pixels, but shadow casting will require more steps for finer resolutions.

The weakness of DEMs is that they do not represent vertical surfaces, except by implication. So, although they are much used for landscape modelling and hydrological studies, they are not of much use for civil engineering structures, where vertical or overhung surfaces are likely. The smearing of edges, which effectively knocks them a little bit out of true vertical is a partial solution, and allows some low level of detail to be stored, such as the difference in

lighting on the upper and lower parts of a wall. But this is crude, and must put some limits on what can be done with the light inter-reflection algorithm.

NIH Image itself has many limitations and problems. The 8 bit precision, though perhaps appropriate to the approximations in our models, causes all kinds of difficulties. Representing signed quantities (like cosines) is awkward; we use an excess-128 coding, but then the arithmetic operations do not work properly, and the work-arounds are elaborate to program. There is some capability to store real numbers (it is needed by the FFT command), but it is not fully supported. Convolutions using kernels which sum to zero, and can hence yield negative results, do not operate properly, and we have had to use shifts and add/subtract to implement the Sobel filter. The macro language, though reliable, has no debugging support, and omits some important features of Pascal, such as user-defined arrays, and structured data-types.

There are alternative image-processing packages, such as MATLAB, which overcome these problems, and may ultimately be more attractive, certainly for the more complex algorithms. MATLAB uses double precision, so need at least eight times the memory, and is correspondingly slow. However its matrix-manipulation language effectively gives you an algebra of images, and a very direct way of encoding the sort of algorithms we have been describing. NIH Image, because of its 8-bit limitation, is very fast, and very economical of memory. It is also very interactive – every command and most macro statements produce visible results on the screen, while in MATLAB you have to program explicitly any display that is needed. However, one sometimes wishes that some of the NIH Image display routines could be disabled while a macro is running.

## CONCLUSIONS AND FUTURE WORK

The use of image processing for simulating environmental performance on the urban scale has proved to be unexpectedly effective. Though we have not completed an urban equivalent to the LT Method for analysing building energy consumption, it seems that most of the important problems have solutions.

A great deal more could be done to extend the modelling capability, using multiple images, rather than simply a DEM. If the ground surface is complicated, separate ground and building DEMs would be necessary. Additional channels could be used to classify surface types (vegetation, water, roads, buildings), and to provide more information about buildings, of which the number of stories would be most valuable, followed by information about glazing, albedo, and materials generally. In the end, this would amount to a raster-based Urban Information System, highly adapted for examining the relationship between urban texture and environmental performance, but suitable also for other tasks where built form is important, such as radio propagation analysis.

The study of urban texture is only just beginning, and it is natural to expect image-processing to be of importance in characterising texture. The unexpected result of this investigation is that it may also be a fruitful way in which to examine the environmental correlates of texture.

## ACKNOWLEGEMENTS

To the Project ZED team, for case-study material, to Nick Baker for his concept of Porosity and especially to Carlo Ratti for many discussions, and the coding of some of the algorithms.

## BIBLIOGRAPHY

Baker N V and K Steemers, “The LT Method 2.0, An Energy Design Tool for Non-Domestic Buildings”, RIBA 1995.

Jain A K, “Fundamentals of Digital Image Processing”, Prentice-Hall 1989.

Martin Centre for Architectural and Urban Studies (Coordinators), “Project ZED: Towards Zero Emission Urban Development - The Interralationship between Energy, Buildings, People and Microclimate”, Final Report to the European Commision, DG XII, Contract RENA-CT94-0016: Cambridge, The Martin Centre 1997.

National Institute of Health “NIH Image (Version 1.61)” Software and Manuals down-loadable from http://rsb.info.nih.gov/download.html.

Page J K (ed) “Prediction of Solar Radiation on Inclined Surfaces” Dordrecht, Reidel Publishing Company, 1986.

Sillion F X and C Puech, “Radiosity and Global Illumination”, Morgan Kaufmann 1994.

Webster C J, “Urban morphological fingerprints” in *Environment and Planning B: Planning and Design* (22) 1995.

## APPENDIX

The following is an example NIH Image macro, to compute the shadow volume for a complete image.

```
Function ShadowVol
(inpu:integer;sx,sy,sz:real):integer;
{compute shadow volume }
{inpu - pid of input DEM}
{sx,sy,sz -  vector pointing at sun}
{expects title, scale, unitz, w, h }
{method:  offset built volume along the}
{azimuth vector, reduce height according to
alt angle, max with volume so far}
var
dmax,i,imax:integer;
 shad,offs:integer;
dx,dy,dz,step:real;

begin
{window to accumulate shadow vol}
MakeNewWindow(concat(title,'.shadvol'));
shad:=PidNumber;

{window for offsetting}
MakeNewWindow(concat(title, '.ofs'));
offs:=PidNumber;

{increments for each step}
{larger of x and y must be 1 pixel}
dx:=abs(sx);
dy:=abs(sy);
if dx>dy then step:=1/dx else if dy>0 then
step:=1/dy else step=1;
if dx>dy then dmax:=w else dmax:=h;
dx:= -step*sx;
dy:= -step*sy;
dz:= -step*sz*scale/unitz;

{number of iterations}
imax:= trunc(-255.0/dz);
if imax>dmax then imax:=dmax;
if imax<1 then imax:=1;

{main loop}
for i:=1 to imax do
begin
      ChoosePic(inpu);
      SelectAll;
      Copy;
      {input image to clipboard}
      ChoosePic(offs);
      MakeRoi(i*dx,i*dy,w,h);
      {makes an offset selection region}
      Paste;
      {and paste image into it}
      KillRoi;
      AddConstant(round(i*dz));    {reduces
levels in offset copy}
ImageMath('max',shad,offs,1,0,shad);
      {take max and store in shad}
end;
ShadowVol:=shad;
ChoosePic(offs);
Dispose;
end;
```